6.868 Final Project

Malcom Gilbert

Sunny Long

Music is pervasive throughout society. Some estimate the age of music to be greater than 50,000 years, dating way back. Through these millennia, we often pair music types with different occasions. For instance, at a wedding, we think that it is appropriate to play Ode to Joy and Pachabel's Canon - songs tied with joy - whereas, at a funeral, we may play a slow procession. Another item that is closely tied to different occasions are speeches. At a wedding, toasts are given in high excited voices, while at a funeral, eulogies are given in slow deliberate phrases.

In this paper, we would like to present a theory on the close ties between language and music interpretation. More specifically, we would like to show that there are many innate elements of music that we are born with (many of which are tied to emotion), and that we use many of these elements in language (speech) to evoke emotions. To do this, our paper will be laid out in four sections. The first section will give some background on music and speech and make the claim that speech and language are no more than a subset of a greater realm of music (to be defined later). The second section will present the model in which we connect both language and music to emotion. It will elaborate on the born-with and developed portions for these connections. The third section will introduce an experiment we performed to verify some innate musical elements to humans (that carry over to language). Finally, in the fourth section, we will conclude with further empirical evidence of the close ties between music and speech processing as they relate to emotion.

Before we start, we would first like to establish some definitions that we are working with. As (Brown, Merker, and Wallin 2000) write in *The Origins of Music*, "The question what is music? is one that has no agreed-upon answer." Any quality that one can bring up to describe "music" can be refuted by a genre of music or even particular compositions. For instance, John Cage's *4'33"* is a piece crafted in 1962 that includes simply four minutes and thirty-three seconds of silence. In it's first performance,

Cage wrote, "In a situation provided with maximum amplification, perform a disciplined action." Despite "music" lacking a concrete unified definition, in our paper we will describe any sound that includes an aspect of tempo and pitch, where tempo is defined as the speed of the sound, pitch is how high or low the sound is. Once we define music, we can turn to language and speech. Here, again, we will employ a relatively lax definition: spoken language is any form of sound produced from one person with the intent of sending information (that is understood by more than just the speaker).[1]

With these definitions, we would like to present a model where language exists wholly as a subset of sound, and what we mean by this is that everything that is classified as language, can be classified as music (as we defined), but not everything that is music can be defined as language. We justify this in two steps, first showing that anything classified as language belongs under the umbrella of music; and then show that there exists items in music outside the realm of language.

For our definition of language - as it is sound emitted, it will have a natural pitch and tempo. People or animals alike produce sound (communication) and this sound takes on a pitch, or from a scientific level, just a frequency. Tempo is simply how closely the frequencies are interlaced, or how closely one sound follows another. Language, with these properties falls squarely under the realm of music. Now, consider the other way around - music, simply sound with pitch and tempo, may not be conveyed as language since the additional aspect in language is one of understanding by the counterpart. If I make up sounds, without comprehension from those around me, it cannot be construed as a language as we define it.

The last clarification we would like to present is in reference to the introduction to this paper, in which we establish that different types of music and speech are paired (and customary) to different occasions. Many of these pairings are decided upon by the type of occasion (happy, festive, sad, etc.). To further discuss this, we must first define what we mean by "happy", "sad", and the other emotions. In doing so,

[1] These definitions are open to debate and are simply supplied as an attempt to further our argument, as without a basic definition, it would be hard to move forward. We will focus primarily on the spoken language and speech itself as opposed to the many nuances of language and representations used to adapt to the lack of a representation in words.

we will utilize ideas from Minsky's "The Emotion Machine". In his book, Minsky describes emotion by utilizing the Critic-Selector model of the mind. Plainly, when an event occurs, a corresponding Critic recognizes the problem type and then activates a Selector which will then evoke a certain "Way to Think". We can now view these emotions simply as the result of a "large-scale cascade" in which many of our active Critics are either turned off or changed significantly in their level of expression (Minsky 2006). To reiterate our previous point, both music and language alike have been known to evince different emotions. By building upon Minsky's model of emotions, we hope to develop our own model of how music and language are related.

From the Critic-Selector model, we can understand how both music and speech tie in to emotions evoked. Our critics for tempo and pitch will activate the necessary selectors, which then cascade, leading eventually to the emotions we feel. For example, when we listen to a slow funeral procession, our critics for pitch will note that the tones are low while the critics for tempo can tell that it is a very slow tempo. These then activate the needed selectors, eventually allowing us to feel "sad." As a parallel, in speech, our critics operate similarly, when we hear a eulogy, our pitch critics will detect low tones, and tempo critics will sense a slowness to the speech. These then activate selectors which will lead to us also feeling "sad." Below, we will explore the argument that these critics are shared between the two processes of interpreting music and speech.

Now, if we can come to accept that language and speech are no more but a subset of music, then we would like to present our theory that closely bind many of the inner workings of music and language. Borrowing from Winston's (1975) theory on a four-level scheme for vision as adapted by Minsky (1981) to apply to music:

Feature-Finders listen for simple time-events such as notes or peaks or pulses.
Measure-Takers notice certain patterns of time-events like 3/4, 4/4, 6/8.
Difference-Finders notice that figure X is like figure Y, but higher by a fifth.
Structure-Builders notice that three phrases form a regular "sequence."

we construct a similar scheme for music and language. Our theory is music and language share much of the same mechanism and agents that connect them to emotion. We will draw parallels between speech and music and show how their mechanisms that tie them to emotion are similar and shared.

The see these parallels we will also need to utilize Minsky's 6-level model of the mind as well as the models already discussed above. To begin, we will simply state our hypothesis. We propose that Feature-Finders, Measure-Takers, Difference-Finders, and Structure-Builders are all representative of Innate or Instinctive Reactive Critics; the emotional cascades triggered by the brain are created from Learned Reactive Critics; language is distinguished from music by the set of Learned Reactive Critics that process it. Our theory then states that for critics corresponding to speech and those corresponding to music, they both cascade to at least a shared set of Critics (the Innate or Instinctive Reactive Critics).

We can see initially that our classification on the Instinctive level are due to the automatic responses we hear from loud noises, repetitive sounds, and harmony. It is sufficient to say that loud noises are innate due to the common head-jerking reaction towards a baby's cry. For Measure-Takers and Structure-Builders, we can look at phenomenon such as when we notice that our foot is tapping a rhythm to some barely audible sound that we had not previously been aware of. Finally, for Difference-Finders, we only need to look at the pervasiveness of harmony and how everyone can recognize differences between major and minor scales, although they may have difficulty verbalizing them. Finally, this theory makes intuitive sense as the brain needs to break down the sound into it's constituent parts before it would be able to extract any sort of useful information from it.

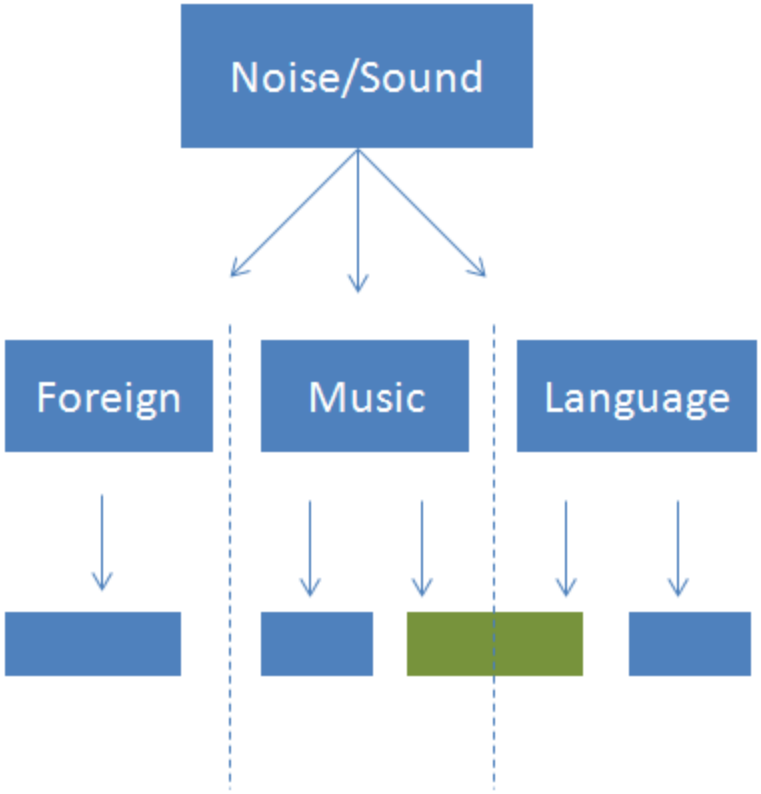

Figure 1: The Critic Selector model showing that music and language share a large portion.

From Figure 1, our theory suggests that the green block, shared by both music and language is the direct link to emotion. This green block includes feature finders that identify notes and pitch. Measure takers can then look for different tempos. Finally, as Minsky (2005) writes, our *Storytellers* will help us link to *abstract emotional scripts*. Though Minksy (2005) only describes this function for music, we would like to make the case that this is shared by Language interpretation as well.

Apart from the meaning of the words used, how words are expressed sometimes mean more than the words themselves. The pitch and tempo contribute to the meaning of a phrase. For instance, in English, a phrase as innocuous as "Jack went to school today" could carry on different meanings:

- Jackwentotschooltoday!!!!!
- Jack went to school today?
- Jack went to school today.

The first instance is a rushed phrase, frenzied, which carries an air of excitement. The second, with a small inflection upward in pitch at the end turns the phrase into puzzlement. The last, with neutral tones simply states a fact. There are numerous variations that can imply different things with such a simple phrase. The use of pitch and tempo in everyday speech contribute enormously to meanings of everyday speech. In our model, we accredit these small changes in pitch and tempo to people's innate music comprehender. We believe that people are born with certain understandings of pitch and tempo. We would characterize this under "Instinctive Reactions" whereas other aspects of language and music go to "Learned Reactions."

To show that people are born with certain bases for pitch and tempo recognition and differentiation, we developed an experiment to test this idea. We realized that in today's society, music (remember, tempo and pitch?) is almost mostly standardized. Pianos with their 88 keys are divided into white and black keys - only accounting for 88 notes. Thus, there are a plethora of notes that are often not played. Consider an A and a B. We have A-sharp, B-flat, but how about something between A and A-sharp? What about moving up a quarter of a note instead of a half? Thus, for our experiment, we will build chords and scales with notes that are not often heard in music. We have created a program that produces these "off" chords and scales while varying tempos. Through playing these "unknown" notes, we hope that the people that hear them can still listen and distinguish when a chord is off, or when a scale is wrong. See Appendix A for our code documentation.

Results

After running our program and testing on 18 students, we found some very interesting results. For all of our experiments, first played the individual notes of the chord and then we played the full chord. In our files, we repeated this eight times. This was mostly due to limitations of the library that we were using. We simply needed the audio files to be long enough so that our program could perform a decent analysis. In our program, we allowed the user to input a value between 0 and 1 that would modulate the

pitch of the note by that corresponding amount. That way, we could simulate the "unknown" notes as described earlier.

For our first experiment, we simply altered a major fifth scale rooted on C by changing the pitch of each note in the chord and then mixing the resulting notes together. The results of this experiment were very intriguing. At the beginning of the experiment, many users had trouble simply deciding if the chord still sounded correct. They would not commit to an answer and sometimes needed to hear the audio played again (after the 8 iterations of hearing the chord). One major reason for this is one of our sources of error for the experiment. The API that we were using to manipulate audio resulted in a chord that was softer than an individual note after mixing and was not as clear as the individual notes. By 'not as clear', we mean that it was not as loud and had a hint of the same muffled effect as when you put your hand over your mouth when you try to speak. Also, the fact that they had trouble deciding is consistent with our model of how the mind processes music. Our users are simply using their Deliberative Critics to create a new Selector which activates a Way of Thinking about this new somewhat "muffled" chord type. This theory is given more weight since, as the experiment progressed, users were able to decide about the sound of the chord quicker.

Once the users had listened to our new manipulated chords enough, they all were able to discern that shifting the pitch of the individual notes did not have an impact on the perception of the chord. More plainly, the major fifth chord still sounded like a major fifth chord even after shifting the notes by an unusual amount.

We then tried different types of chords. For three note chords, we tried a minor chord, an augmented chord, and a diminished chord. User performance on the minor chord was analogous to what was seen with the major chord. Performance on the diminished chord and augmented chords, however, suffered from the same deliberation difficulty as discussed before. Initially, users were unable to commit to whether the chord was still diminished or augmented. This time, though, the effect was much shorter. In our model of the mind, we can simply explain this by thinking about the Selector that was processing the

previous major and minor chords. In order to now process diminished and augmented chords, that Selector only needs to be mutated a bit with the information specific to diminished and augmented chords. It can then create a new Way of Thinking that can accommodate our new chords.

We also briefly tested this pitch shift on four note chords. We only briefly tested this because our users had significant difficulty determining if the chords had the correct sound. We would like to explain this again based on our model. For the four note chords we tried to test (dominant seventh, major seventh, minor seventh, major/minor seventh, etc...), there had not been a lot of previous exposure to these chords. Without this exposure, the mind has not been able to create Learned Reactive critics that can efficiently recognize the chords and evoke the requisite Selectors to active a good Way of Thinking about the chord.

Next, we changed one note of the chord and tried to analyze the results. This experiment went a lot smoother than the previous. When the chord was dissonant, the users could immediately tell that something was amiss. There was, however, an interesting side effect. We did receive many false negatives. For instance, when the chord was still a major chord but a note was shifted by an octave, many of our users thought that the chord sounded wrong. We could look at this in two different ways. One way would be that our users fail to accurately categorize the chord. The other would be that the users were able to distinguish that the pitches were not the same "distance" apart from each other. With this interpretation, we can say that the Difference-Finders have been able to distinguish the relative position of the notes, but that the current Way of Thinking cannot make the link to classify the sequence of nodes correctly.

We also changed the order in which the notes were played to the user before playing the chord. Instead of playing C-E-G for a major fifth chord, we would randomly shuffle these notes into something like E-C-G. We then questioned our users about the ability to hear the chord in their head before it was finally played after the shuffled sequence of notes. Our users found this particularly challenging even for the simplest of chords. We can view this as a result of the suppression of our Structure-Builder Critics.

These critics are responsible for recognizing the sequences of notes. Since - in our model - these are innate, it makes sense that users could no longer identify the chord immediately. We have disrupted the regularity of our sequence so it is not processed via the same set of Critics.

Finally, we manipulated the tempo of all of our chords. This was the least interesting experiment as it didn't really alter the pitches in the chord. The only edge cases came when the chords were slowed down or sped up by extreme values. To explain this, we simply note that our Feature-Finders are simply unable to distinguish between the notes and may treat them as the same note if they are played fast enough in succession. When the notes are played too slowly, we can think that Critics to recognize the problem type have already fired before the entire event has concluded thereby misclassifying the problem type.

Conclusion

From the results, we see that there is some credibility to the theory that there are some innate aspects to music that humans are born with. As we grow, we develop new associations (language itself included) with music. We argue that those parts that are innate to us (born with) are shared between both the music and speech critics. We first draw similarities between music and speech, showing that most elements of music are represented in speech as well. Then we show, through our experiment, that the roots of these shared elements of music (pitch and tempo) are items that we are born with. Thus, we can conclude that our theory in which these shared critics exist holds water. Furthermore, we can draw from recent studies, Macquarie (2012) and Thompson (2013) both study the effects of certain types of amusia on speech understanding - focusing on the emotional underpinnings. These papers present experiments that brought together people with congenital amusia and a control group and let them listen to 96 spoken phrases. These phrases were semantically neutral, but prosodic cues communicated six different emotional states. Their results show that the congenitally amusic individuals were significantly worse at decoding emotional prosody. The authors further write that these results

> "*lend support to early speculations by Darwin, elaborated upon by several contemporary theorists, that emotional communication is a fundamental link*

between these domains and reflects their common evolutionary origin."

These results are supportive of our hypothesis that there are shared critics between music and speech processors. We have shown that there are innate critics shared between music and speech interpretation that are linked to emotions. Our results are encouraging and suggestive of people being born with these traits.

In this paper, we mainly discuss the innate common bases for music and speech interpretation that people are born with. However, once people start developing into adults, these two paths diverge - new critics specific to just music and to just speech develop. As Minsky does, we will refer to Curtis Roads remarks: "Every world above bare survival is self constructed; whole cultures are built around common things people come to appreciate." Thus, as society comes to appreciate different elements of music in different ways, our critic-selector model adapts as well, associating some higher level ideas from both music and speech to different things, but only from the bases that we are born with.

References:

Winston, P. H. 1975. "Learning Structural Descriptions by Examples." In P. Winston, ed. 1975. *Psychology of Computer Vision.* New York: McGraw-Hill, pp. 157-209.

Minsky, Marvin 1981. “Music, Mind, and Meaning.” Computer Music Journal, Fall 1981, Vol. 5, Number 3.

Minsky, Marvin. *The Society of Mind*. Simon and Schuster, New York. March 15, 1988. ISBN 0-671-65713-5

Minsky, Marvin (2006). *The Emotion Machine*. Simon & Schuster. ISBN 0-7432-7663-9.

Macquarie University, *Research finds emotions fall on tone-deaf ears*. October 30, 2012.

Thompson, W., Manuela, M., Steward, L. *Reduced sensitivity to emotional porsody in congenital amusia rekindles the musical protolanguage hypothesis*., Proceedings of the National Academy of Sciences of the United States of America. February 12, 2013. vol. 110 no. 7 2677.

Khan, Amina. *Study: Music, language's common evolutionary roots lie in emotion*. Los Angeles Times, October 30, 2012.

Appendix A

Code documentation is written in the python docstring format under each method.

Code is also on github along with audio files used: https://github.com/mjgil/6868

main.py

```
import argparse
from remix import REMIXES

def parse_args():
    """
    Parses the command line arguments using the argparse python module

    Use -h to see the options.
    """
     print 'parsing arguments...'
     description = "Manipulates 3 and 4 note chords by changing pitch, tempo, and
arrangement of notes"
     chord_type_help = 'Specify either three_note or four_note'
     base_chord_type_help = 'Name of base chord that you want to use (ex. major_fifth)'
     remix_type_help = 'What to change in the chord (one_note_pitch_shift, change_tempo,
change_note_order, all_notes_pitch_shift)'
     remix_amount_help = 'Amount to change the remix type (Optional - Value between 0 and
1)'

     parser = argparse.ArgumentParser(description=description)
     parser.add_argument('chord_type', help=chord_type_help)
     parser.add_argument('base_chord_type', help=base_chord_type_help)
     parser.add_argument('remix_type', help=remix_type_help)
     parser.add_argument('remix_amount', help=remix_amount_help, nargs='?')
     return parser.parse_args()

def main(args):
     """
    Main entry point of the application.
```

```
    Gets the appropriate Remix class based on the passed in command line
    arguments. Initializes the class if it is not None and then proceeds
    to process the audio.
    """
     print "entering main..."
     remix = REMIXES.get(args.chord_type, None)
     assert remix is not None, 'Chord Type Invalid'

     remix = remix(args)
     remix.process()

if __name__ == '__main__':
     args = parse_args()
     main(args)
```

remix.py

```
import os
import statics
import dirac
import random
from pyechonest import config
from echonest.remix import audio,modify
config.ECHO_NEST_API_KEY="UJGOWAOWXLAR4SBR9"
REMIXES = {}

def is_number(s):
  """
    Input: s - can really be anything

    tries to cast it to a float
    if it works return True
    else return False
```

```
    """
    try:
        float(s)
        return True
    except ValueError:
        return False


class BaseRemix(object):
    def __init__(self, args):
        """
        Input: args -- object created from parsed command line arguments

        Initializes the basic remix class.

        First, it forms the path for the input file from the
        parameters that were inputted to the program.
        If the file cannot be found, it exits the program.

        Next, parses the remix_amount parameter or sets it to a
        random float between zero and 1. If the inputted value
        is not a number or is not between 0 and 1, it exits the
        program.

        Finally, it sets the remix_type to be a class variable
        and creates the output file path. It also creates the
        output folder if it does not already exist.
        """

        self.input_file = os.path.join(statics.input_path,
                                       args.chord_type,
                                       args.base_chord_type) + '.wav'
        assert os.path.exists(self.input_file), 'Chord Music File Does Not Exist'

        self.remix_amount = args.remix_amount or random.random()
        assert is_number(self.remix_amount), 'Remix Amount Must Be A Number'
```

```
    self.remix_amount = float(self.remix_amount)
    assert (0.0 <= self.remix_amount <= 1.0), 'Remix Amount Must Be Between 0 and 1'

    self.remix_type = args.remix_type
    self.output_folder = os.path.join(statics.output_path,
                                      args.chord_type,
                                      args.base_chord_type)
    if not os.path.exists(self.output_folder):
      os.makedirs(self.output_folder)

    remix_str = '%.2f' % self.remix_amount
    output_file_name = "_".join([args.remix_type, remix_str]) + '.mp3'
    self.output_file = os.path.join(self.output_folder, output_file_name)

  def process(self):
    """
   Pre-processes the inputted audio file (self.analyze())

   Then calls the corresponding function passed in through
   the remix_type argument.
   """
    self.analyze()

    fn = getattr(self, self.remix_type)
    assert hasattr(fn, '__call__'), 'Invalid Remix Type'
    fn()

  def change_tempo(self):
    """
   Changes the tempo of the beats in the chord

   Gets the beats from the self.beats list then
   changes the tempo as specified by the remix_amount
   that was either generated on initialization of this
   class or passed in through the command line
   """
```

```
  collect = []
  for x in range(8):
    for beat in self.beats:
      ratio = self.remix_amount + 0.5
      beat_audio = beat.render()
      scaled_beat = dirac.timeScale(beat_audio.data, ratio)
      ts = audio.AudioData(ndarray=scaled_beat, shape=scaled_beat.shape,
                           sampleRate=self.audiofile.sampleRate,
                           numChannels=scaled_beat.shape[1])
      collect.append(ts)

  out = audio.assemble(collect, numChannels=2)
  self.encode(out)

def change_note_order(self):
  """
 Changes the order of the notes played before a chord

 Gets the notes that make up the chord (all of the beats
 except for the last one) and randomly shuffles them.

 Next, it adds the chord back in and renders the audio out
 """
  collect = []
  beats = self.beats[:-1]
  random.shuffle(beats)
  chord = self.beats[-1]
  beats.append(chord)
  for x in range(8):

    for beat in beats:
      beat_audio = beat.render()
      scaled_beat = dirac.timeScale(beat_audio.data, 1.0)
      ts = audio.AudioData(ndarray=scaled_beat, shape=scaled_beat.shape,
                           sampleRate=self.audiofile.sampleRate,
                           numChannels=scaled_beat.shape[1])
```

```
      collect.append(ts)

  out = audio.assemble(collect, numChannels=2)
  self.encode(out)

def one_note_pitch_shift(self):
  """
 Picks one of the notes at random from the chord and
 modulates it based on the remix_amount parameter

 Gets all of the notes of the chord from self.beats.
 Picks one of the notes at random.
 Shifts that note by the remix amount.
 Then combines all of the notes together to generate
 a new chord from the notes.
 """
  random_index = self.get_random()
  for x in range(8):
    beats = self.beats[:-1]
    beat_list = []
    for j, beat in enumerate(beats):
      shift_ratio = 1
      if j == random_index:
        shift_ratio = shift_ratio * self.remix_amount
      new_beat = self.soundtouch.shiftPitch(self.audiofile[beat], shift_ratio)
      self.out_data.append(new_beat)
      beat_list.append(new_beat)

    new_beat = self.mix_beat_list(beat_list)
    self.out_data.append(new_beat)
  self.encode(self.out_data)

def all_notes_pitch_shift(self):
  """
 Picks modulates the notes of the chord by an amount
 specified from the remix_amount parameter
```

```
    Gets all of the notes of the chord from self.beats.
    Shifts the notes by the remix amount.
    Then combines all of the notes together to generate
    a new chord from the notes.
    """
     for x in range(8):
       beats = self.beats[:-1]
       beat_list = []
       for beat in beats:
         shift_ratio = self.remix_amount
         new_beat = self.soundtouch.shiftPitch(self.audiofile[beat], shift_ratio)
         self.out_data.append(new_beat)
         beat_list.append(new_beat)

       new_beat = self.mix_beat_list(beat_list)
       self.out_data.append(new_beat)
     self.encode(self.out_data)

def encode(self, out):
  """
    Param: out - output audio data

    After all of the audio processing is done,
    renders the newly created audio to the appropriate
    output directories.

    Sends one to the path specified in statics.py and the
    passed in command_line parameters.

    Sends another to the three_note or four_note audio/sample
    directory named 'previous.wav'. This enables us to chain
    together our audio effects easily.
    """
     out.encode(self.output_file)
     prev_path = os.path.join(statics.input_path,self.Name,'previous.wav')
```

```
    out.encode(prev_path)

  def analyze(self):
    """
    Uses the echonest remix api to analyze the input_audio file.
    Saves the information in class variables.

    Finally calls get_beats() to handle the different types of
    chords supported.
    """
    self.soundtouch = modify.Modify()
    self.audiofile = audio.LocalAudioFile(self.input_file)
    self.bars = self.audiofile.analysis.bars
    self.out_shape = (len(self.audiofile.data),)
    self.out_data = audio.AudioData(shape=self.out_shape, numChannels=1,
sampleRate=44100)
    self.get_beats()

  def get_beats(self):
    """
    Implemented in subclasses

    Iterates over the list of bars found through the analyze method
    If the length of the bar matches the number of notes expected,
    4 for a 3 note scale (3 individual notes and then the chord) or
    5 for a 4 note scale then we save the beats in the correct order
    in self.beats.

    This method is a bit weird because the echonest remix api doesn't
    always do the best job in sorting out the beats. To mitigate this
    we made sure that our audio files followed a certain pattern and
    then coded to match our pattern.

    All of our audio files play the notes in the scale first and then
    the chord at the end. This is repeated 8 times so that the remix
    API can do a good job with it's analysis.
```

```
    """
     raise NotImplementedError

  def mix_beat_list(self, beat_list):
    """
    Implemented in subclasses

    Args: beat_list - a list of beats found through the echonest remix API

    Mixes the beats together using the audio.mix method of the remix API.
    There is a megamix but we found that mixing the beats individually
    resulted in higher quality audio.
    """
     raise NotImplementedError

  def get_random(self):
    """
    Implemented in subclasses

    Gets a random integer in the range of available notes. 0-2 for the three_note
    class and 0-3 for the four_note class. This is used to pick a random note to
    shift in the one_note_pitch_shift method.
    """
     raise NotImplementedError


class ThreeNoteRemix(BaseRemix):
  Name = "three_note"
  def __init__(self, args):
    super(ThreeNoteRemix, self).__init__(args)

  def get_beats(self):
    # for docs see method definition in BaseRemix
    for bar in self.bars:
      if (len(bar.children()) > 3):
        top = bar.children()[0]
```

```
        chord = bar.children()[1]
        root = bar.children()[2]
        middle = bar.children()[3]
        self.beats = [root, middle, top, chord]
        break

  def mix_beat_list(self, beat_list):
    # for docs see method definition in BaseRemix
    new_beat = audio.mix(beat_list[0], beat_list[1], 0.5)
    new_beat = audio.mix(new_beat, beat_list[2], 0.66)
    return new_beat

  def get_random(self):
    # for docs see method definition in BaseRemix
    return random.randrange(0,3)

class FourNoteRemix(BaseRemix):
  Name = "four_note"
  def __init__(self, args):
    super(FourNoteRemix, self).__init__(args)

  def get_beats(self):
    # for docs see method definition in BaseRemix
    for bar in self.bars:
      if (len(bar.children()) > 4):
        top = bar.children()[4]
        chord = bar.children()[0]
        root = bar.children()[1]
        middle = bar.children()[2]
        middle1 = bar.children()[3]
        self.beats = [root, middle, middle1, top, chord]
        break

  def mix_beat_list(self, beat_list):
    # for docs see method definition in BaseRemix
    new_beat = audio.mix(beat_list[0], beat_list[1], 0.5)
```

```
        new_beat = audio.mix(new_beat, beat_list[2], 0.66)
        new_beat = audio.mix(new_beat, beat_list[3], 0.75)
        return new_beat

    def get_random(self):
        # for docs see method definition in BaseRemix
        return random.randrange(0,4)

# stores subclasses of BaseRemixes in the REMIXES
# dictionary so that they can be read in main.py
for remix in BaseRemix.__subclasses__():
    REMIXES[remix.Name] = remix
```

statics.py

```
input_path = "../audio/sample/"
output_path = "../audio/processed/"
```